

Development Studio 5 (DS-5)

Development Tools for ARM Linux

ARM®

Quick Start Guide

The **ARM® Development Studio 5 (DS-5™)** is a complete suite of software development tools for ARM processor-based systems.

DS-5 includes:

- Highly optimizing C/C++ compiler.
- Powerful debug and performance analysis tools.
- Intuitive and flexible graphical user interface.
- Worldwide, professional technical support.

DS-5 accelerates and de-risks your software development by providing an easy to use, robust, integrated, validated and maintained toolchain.

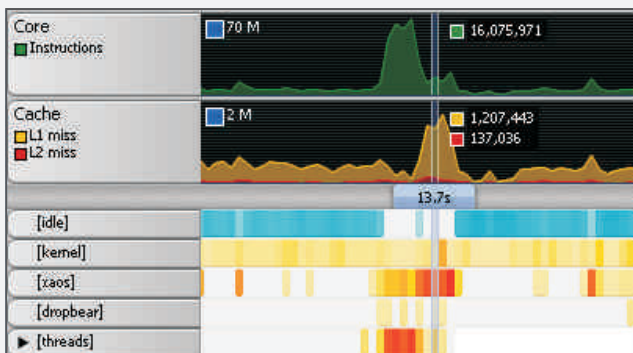
DS-5 covers all the development stages of ARM-based products, from platform bring-up to application profiling, while including a number of ARM Linux specific features.

This Quick Start Guide is intended to help you through your first steps with DS-5, from installation to launching and using each of its components.

What's New!

Streamline performance analyzer

Streamline is a graphical system-level profiling tool with time-based analysis, which enables fast code optimization. It now supports code balancing and thread analysis on Cortex-A9 Linux SMP targets.

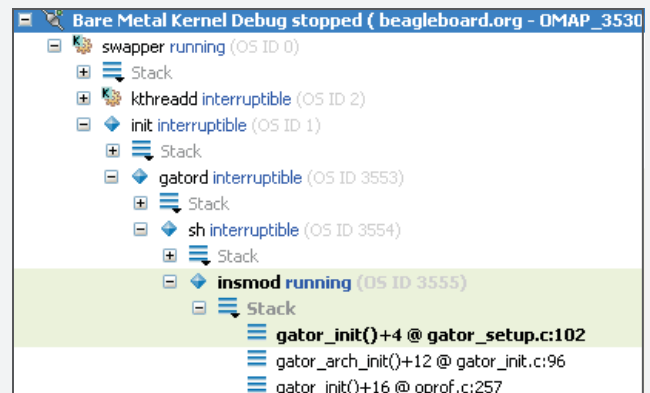


Self	Process	Total	Stack	Process/Thread/Function Name
0.00%	100.00%	0.86%	0	[threads]
0.00%	20.39%	0.18%	0	[thread 4]
0.00%	20.39%	0.18%	0	[thread 6]
15.92%	15.92%	0.14%	0	__addsf3
4.47%	4.47%	0.04%	160	accumulate
0.00%	20.11%	0.17%	0	[thread 2]
15.64%	15.64%	0.14%	0	__addsf3
4.47%	4.47%	0.04%	160	accumulate
0.00%	19.27%	0.17%	0	[thread 3]
16.48%	16.48%	0.14%	0	__addsf3
2.79%	2.79%	0.02%	160	accumulate

Samples	Instances	Function Name
77.93%	5	__addsf3
21.51%	5	accumulate
0.56%	1	[ld-2.11.1.so]

Bare metal, kernel and driver debug

DS-5 Debugger now supports platform bring-up, boot code, and Linux kernel and driver development using a JTAG connection to the target. DS-5 gives access to kernel resources and dynamically loaded modules.



Name	Symbols	Address
/writeable/gnome/gnome-support.so	loaded	0x40027000
/usr/lib/libgtk-x11-2.0.so.0	no symbols	0x40041000
/usr/lib/libgio-2.0.so.0	no symbols	0x40331000
/lib/libstdc++.so.6		0x40000000
/lib/libm.so.6		0x40000000
/lib/libgcc_s.so.1		0x40000000
/lib/libc.so.6		0x40000000
/usr/lib/libgdk_pixbuf-2.0.so.0		0x40000000
/usr/lib/libgdk-x11-2.0.so.0		0x40000000
/usr/lib/libpangocairo-1.0.so.0		0x40000000
/usr/lib/libcairo.so.2		0x40000000
/usr/lib/libpango-1.0.so.0		0x40000000
/usr/lib/libgobject-2.0.so.0	no symbols	0x40798000

Getting Started with DS-5

Downloading

Download DS-5 and a 30-day free, fully featured, evaluation license from www.arm.com/ds5. Alternatively, request a DVD from your local ARM office or tools distributor.

In order to retrieve a license file for your DS-5 serial number, go to <https://silver.arm.com/licensing/>.

DS-5 is regularly updated with new functionality. When you purchase DS-5 you are entitled to technical support and access to free updates for one year. It is recommended that you keep your DS-5 installation up to date.

Running and Licensing DS-5

Most DS-5 functionality can be accessed from its Eclipse environment. **Eclipse for DS-5** is launched from Windows **Start Menu ► Programs ► ARM DS-5**.

Installations on Linux hosts include a script to install/uninstall application menus. Alternatively, launch Eclipse by executing **eclipse** from **<install_directory>\bin**.

When launched, Eclipse prompts you to specify the location of a **Workspace**. This is the directory where your Eclipse settings and projects will be stored. We recommend that you create one or more new directories to be used as DS-5 workspaces.

The first time you run Eclipse, open **Help ► ARM License Manager...** to install your DS-5 license file and unlock the DS-5 functionality.

Welcome Screen

At any time you can open the Eclipse for DS-5 welcome screen by selecting **Help ► DS-5 Home**.

The welcome screen provides links to getting started information such as example documentation, hints and tips, and support resources on the web.

Click on the **Go to the Workbench** link to close the welcome screen and start using DS-5.

Environment Variables

The DS-5 installer does not modify any environment variables on your system. DS-5 only requires that the **path** environment variable points to the **<install_directory>\bin** directory.

This is done automatically if you launch the tools from the **Eclipse for DS-5** or the **DS-5 Command Prompt** items in the Windows Start Menu.

Running the tools from a Linux console requires that you manually set the path environment variable.

System Requirements

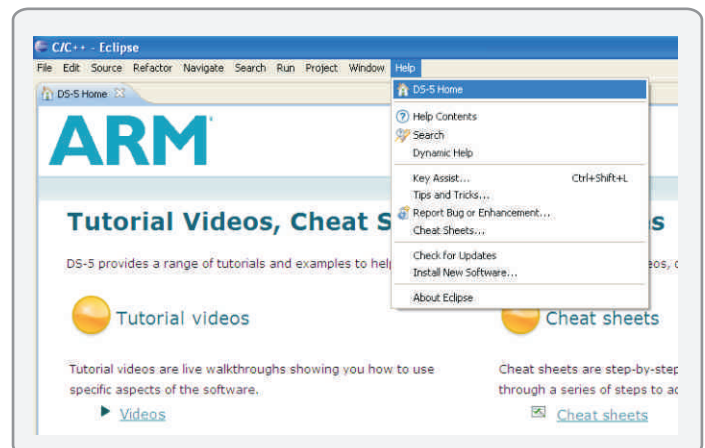
The minimum computer specification to run DS-5 is:

- 2 GHz host processor (dual-core recommended).
- 2 GB of RAM (4 GB recommended).
- 2 GB of hard disk space.

DS-5 supports the 32-bit and 64-bit versions of the following x86 host platforms:

- Windows XP Professional service pack 3.
- Windows 7 Professional and Enterprise.
- Red Hat Enterprise Linux 5 Desktop and Workstation.

Support for Windows Vista service pack 2 is deprecated.



The DS-5 welcome screen links to useful information such as video tutorials, step-by-step guides, examples, and documentation.

Node-locked or floating?

DS-5 node-locked and floating licenses are available for both Windows and Linux hosts.

- **Node-locked** licenses are the best choice for single developers and small teams. They are easy to set up, and ensure that each developer always has the tools available.
- **Floating** licenses are loaded to license servers, and are shared among a team of developers. By using floating licenses you maximise the utilization of each license. In most cases this provides a better return on investment.

Eclipse IDE and Project Manager

Eclipse Perspectives

A typical Eclipse workbench window contains one or more perspectives, each of them including a set of related views, editors, menus, and toolbars.

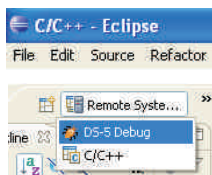
DS-5 includes two Eclipse perspectives:

- **C/C++**: used to import and manage software projects, and to edit source files.
- **DS-5 Debug**: graphical interface for the DS-5 Debugger.

To open an Eclipse view, go to **Window ► Show View**.

The name of the current perspective is shown on the Eclipse title bar.

The buttons on the perspective tab on the top right of the Eclipse window enable switching between perspectives.



Import Example Projects

From the C/C++ perspective, select **File ► Import... ► General ► Existing Projects into Workspace**.

Activate **Select archive file** and browse to `<install_directory>\examples\example.zip`. Click Finish.

This operation decompresses all the DS-5 examples into your workbench directory and opens them in the Eclipse Project Explorer window.

The example projects are also rebuilt automatically with the GCC-based compilation tools included in the DS-5 package.

Any build errors or warnings are displayed in the Eclipse Problems tab.

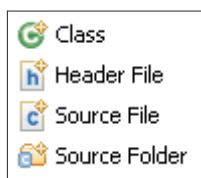
Create and Build Projects

From the C/C++ perspective, select **File ► New ► C or C++ Project**.

Select one of the four project templates in the DS-5 project wizard: executable (Linux application), shared library, static library, or makefile project. This creates a new project with working pre-defined compiler settings.

To add to the project source files based on suitable templates, right-click on the Project Explorer view and select **New ► Source File**.

To make or build a project, select **Project ► Build**.

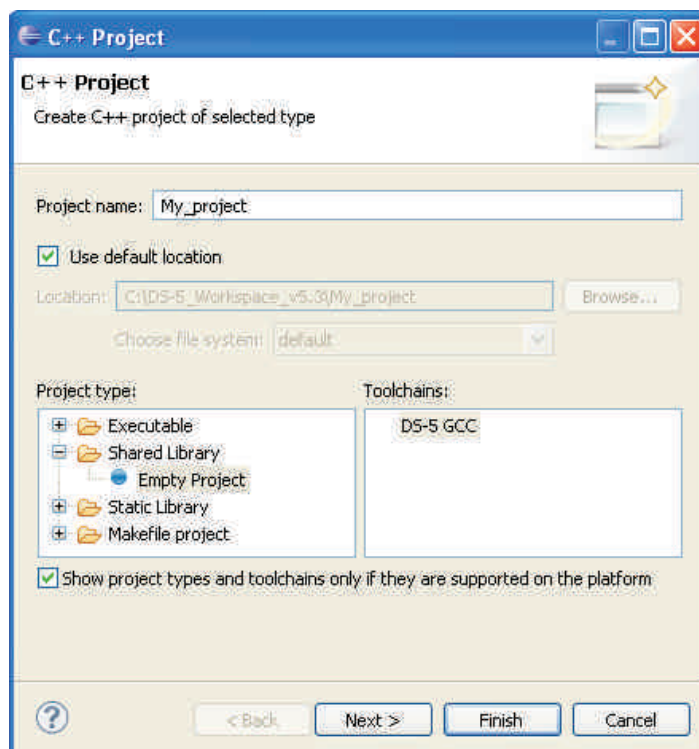


DS-5 Example Projects

DS-5 includes a range of examples to get you started:

- **Distribution** is a complete Linux distribution for the Cortex-A8 Real-Time System Model, which enables application development on this simulation target.
- **Gnometris** is an open source Linux application similar to the game Tetris.
- **Example_library** demonstrates how to create a simple shared library in C for ARM Linux.
- **C++**, **cpp_library**, **cppex** and **cppex_library** are examples of object creation and exception handling
- **Threads** is a simple multi-threaded Linux application that illustrates how the tools handle multiple threads.
- **Xaos** is an example fractal application that includes a pre-generated multi-core Streamline report.
- **Kernel_module** illustrates the kernel awareness features in DS-5 Debugger. Note that it needs to be built on a Linux host against your Linux kernel.

DS-5 also includes a bootloader (**U-Boot**) and a pre-built **SD Card image** to try it on a Beagle board.



Creating new projects in DS-5.

Eclipse C/C++ Perspective

Eclipse provides a powerful IDE (Integrated Development Environment) with flexible window management and C/C++ source editing.

Window Management

Change the position of any view in Eclipse by left-clicking on its tab and dragging it. The new location can be adjacent to existing views or in a tabbed group.

You can **detach** (undock) the view or add it to the **Fast View** bar by right-clicking on the view name.

All detached views can be dragged to a second monitor.

Source Code Editing

The Eclipse IDE includes several features that make it easier to edit C/C++ code, including:

- Syntax highlighting.
- Pop-ups with variable/function information and quick-jump to declaration with **F3**.
- Code auto-completion.
- Templates for source and header files and classes.
- Search in project files.

The ARM Assembler editor also provides syntax highlighting, formatting of code, and auto-completion of labels in ARM and Thumb assembly language files.

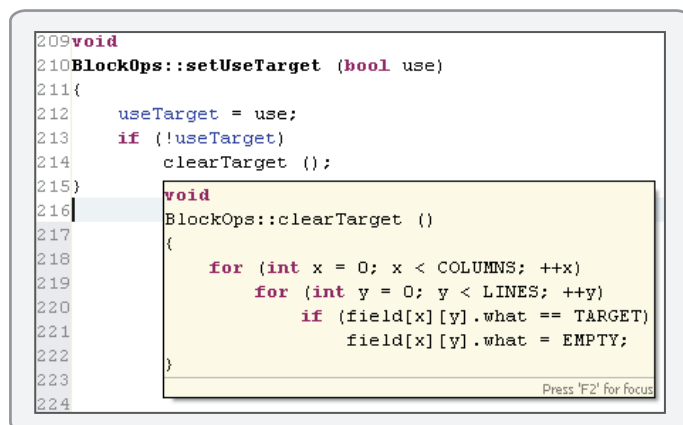
Analysis of Executable Files

Double-clicking on an executable or object file opens the ELF Content Editor plug-in. Alternatively, right-click on the file and select **Open With ► Other ► ELF Content Editor**.

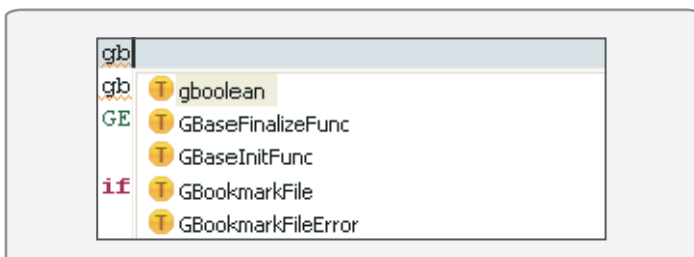
This plug-in enables the static analysis of built or imported images before connecting a debugger to the actual target.

The ELF Content Editor includes several tabs with graphical views for:

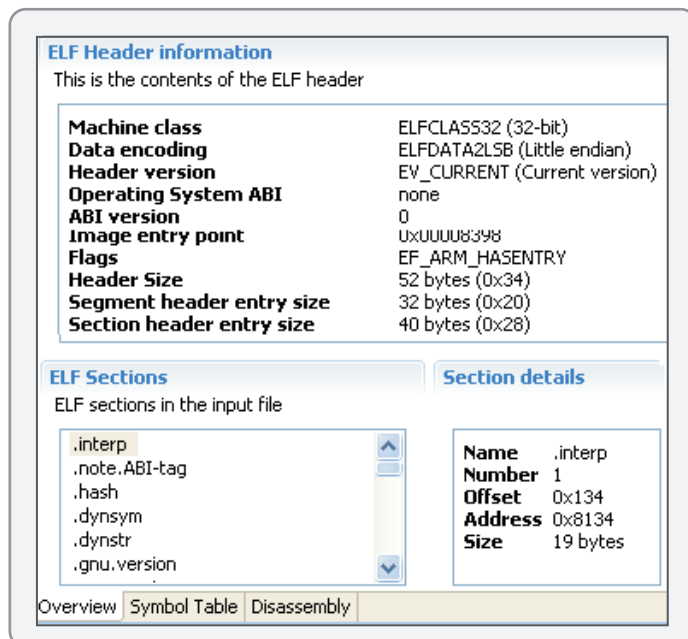
- Overview pane with general information on the image (ELF header) and details for each ELF section.
- Symbol browser with complete symbol list (variables, functions, labels) and symbol properties (type, address).
- Syntax-coloured disassembly view enables you to examine the code generated by the compiler.



Automatic display of function and variable information in the code view.



Source auto-completion. Start typing the name of a class, variable or function and click **Ctrl+Space**.



The ELF Content Editor performs static analysis of object files.

Access to the Target's Linux File System

Connecting to the Target

All you need to debug and analyze Linux applications on an ARM processor-based target is a TCP/IP between the target and the host computer.

DS-5 Debugger launches the **gdbserver** debug agent on the target, connects to it, and uses it to control a running application. DS-5 requires gdbserver v6.8 or later. The installation includes v7.0.

In the case of Real-Time System Models, the debugger connects to one of the UARTs in the SoC model.

Bare metal and kernel development requires a JTAG or Serial-Wire Debug target connection. This is provided by the ARM **DSTREAM™** debug and trace unit.

Accessing the Target's File System

DS-5 integrates a **Remote System Explorer (RSE)** to access the Linux file system on hardware targets that support the Secure Shell (SSH). The target must include a secure shell daemon (**sshd**) and **sftp-server**.

RSE enables the following functionality:

- **SFTP Files View:** Access the target's Linux file system using an FTP connection. Navigate, copy, paste, drag and drop, and edit files on the target.
- **Shell Processes View:** Displays a list of current processes running on the target.
- **SSH Shell/Terminal View:** Opens a shell or a terminal window connected to the target in order to execute Linux commands - for example to run an application.
- **Automated software download:** DS-5 Debugger uses RSE connections to automate the download of images to the target, execute gdbserver, and connect to it.

Configure Remote System Explorer

Open RSE by selecting **Window ► Show View ► Other... ► Remote Systems ► Remote Systems**.

Click on the **New Connection** button . Select Linux and click Next.

Fill the Host name box with the IP address of the target system, or its network name if it is registered on a DNS server. Define a name for the connection and click Next.

In the next configuration steps choose the Secure Shell (ssh.files) option, then click Finish.

Real-Time System Models

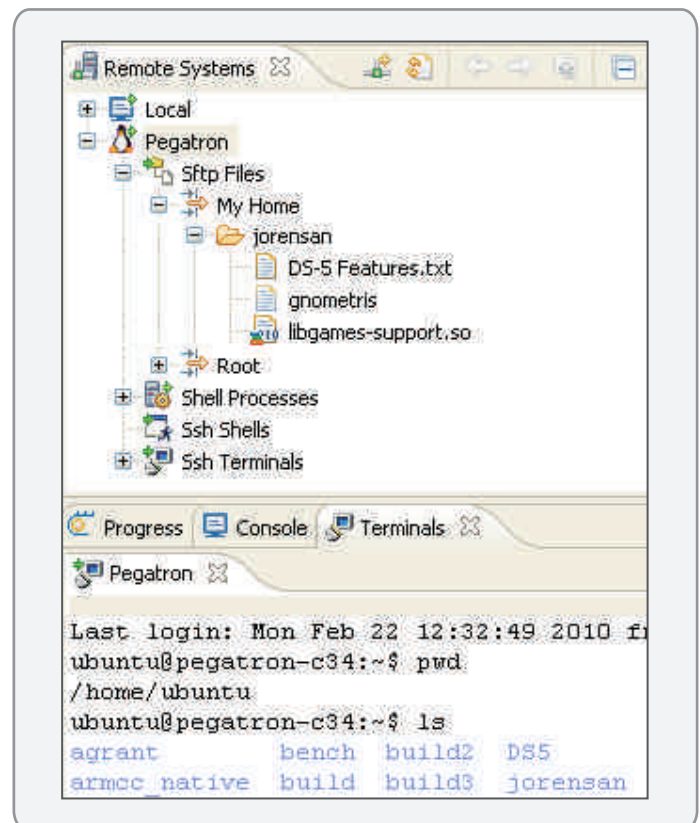
DS-5 includes Real-Time System Models (**RTSMs**) of ARM processor-based SoCs.

An RTSM is a fast simulation model of a complete SoC with a processor, system memory, and peripherals such as keyboard, mouse, UARTs, Ethernet, and LCD.

RTSMs enable ARM Linux application development without any hardware, and deliver simulation speeds of over 250MHz on a typical desktop.

DS-5 includes an example **ARM Linux distribution** ported to the Cortex-A8 RTSM. When launched, the model automatically boots Linux and enters a state in which you can load and debug your applications.

RTSMs are configured to mount a **Virtual File System (VFS)**. The processor running in simulation can access files in the host computer's file system as if they were part of its ARM Linux file system.



Remote System Explorer SFTP and Terminal views.

Starting a Debug Session

Launching DS-5 Debugger

To launch the DS-5 Debugger, click on the down arrow of the Debug button  on the menu bar and click on **Debug Configurations**.

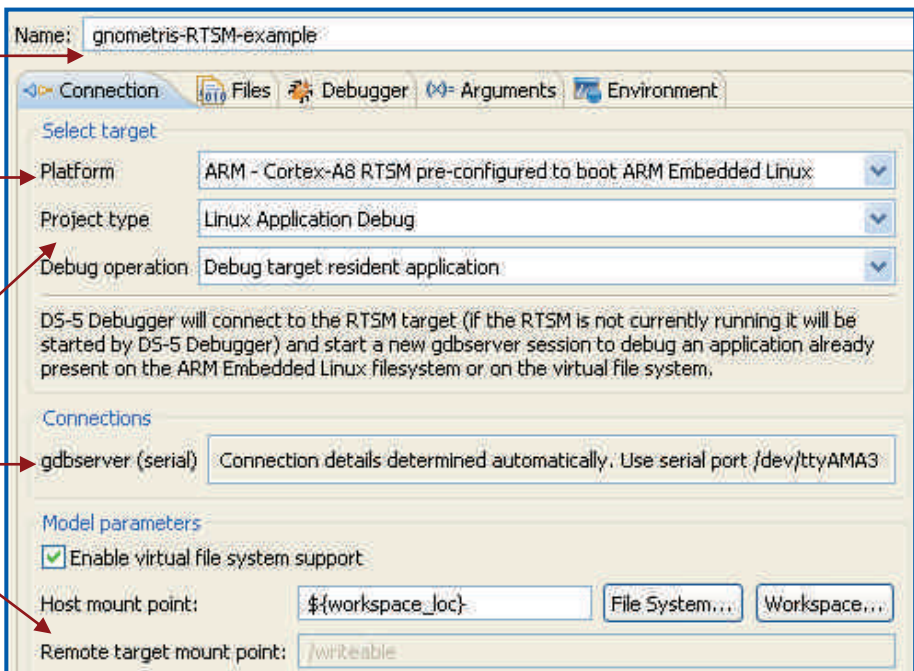
Right-click on **DS-5 Debugger** and select **New** to open the DS-5 Launch Panel.

Import the DS-5 examples into the Workbench to populate the DS-5 Launch Panel with pre-defined DS-5 debug configurations. For instance, try the **gnometris** game on a Real-Time System Model or a Beagle board.

Supported Devices and Platforms

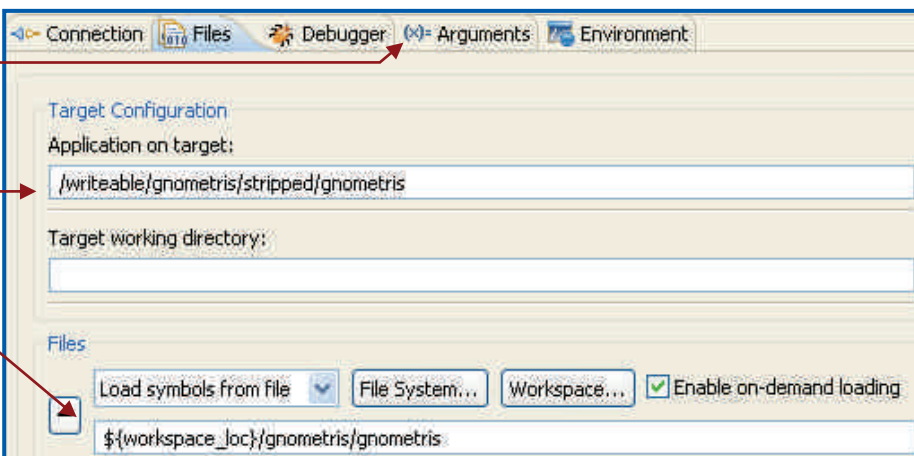
DS-5 provides **pre-configured JTAG connection**, trace, memory map and peripheral registers for catalogue devices. Just choose a device on the Debug Configurations dialog.

A complete list of supported platforms is available on www.arm.com/ds-5



Annotations for the DS-5 Launch Panel:

- Assign a name to the debug connection for later use. (Points to the Name field: `gnometris-RTSM-example`)
- Select a supported device or just a gdbserver connection. (Points to the Platform dropdown: `ARM - Cortex-A8 RTSM pre-configured to boot ARM Embedded Linux`)
- Select bare metal, Linux driver or Linux application debug. (Points to the Project type dropdown: `Linux Application Debug`)
- Target's TCP/IP address and gdbserver port number (not required for RTSM). (Points to the Debug operation dropdown: `Debug target resident application`)
- RTSM can access the Workspace (on the host) as virtual directory `/writeable`. (Points to the Remote target mount point field: `/writeable`)



Annotations for the DS-5 Arguments tab:

- Use the Arguments tab to pass arguments to the application. (Points to the Arguments tab)
- Select application to debug in the target's virtual file system. (Points to the Application on target field: `/writeable/gnometris/striped/gnometris`)
- Symbols on the host must be loaded to do source level debugging. (Points to the Load symbols from file checkbox)

DS-5 Debug Perspective

DS-5 includes a fully featured graphical C/C++ debugger.

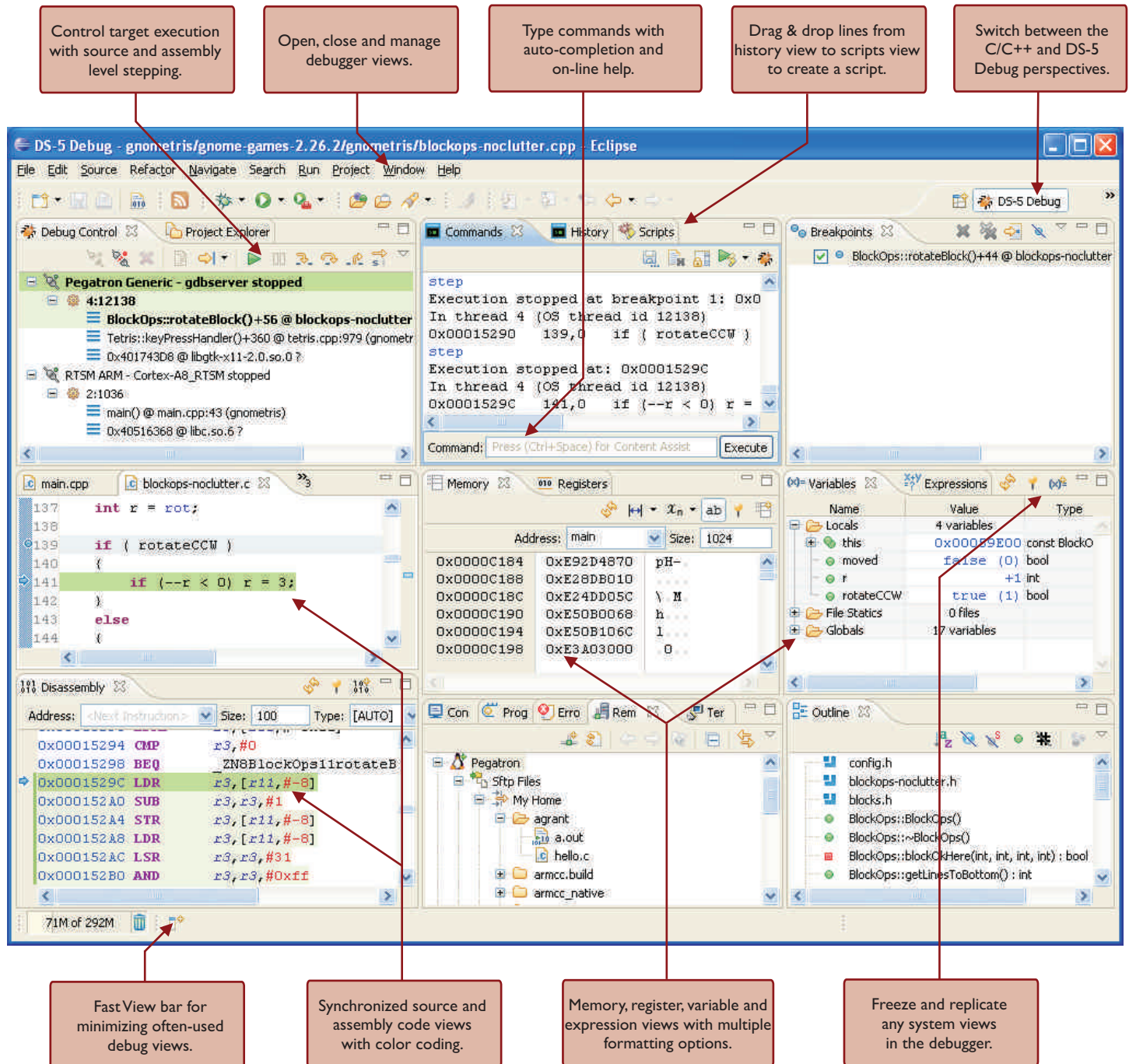
The debugger can be used from the DS-5 Debug perspective, which adds comprehensive system views and processor control mechanisms to the powerful Eclipse IDE. Alternatively, it can be used from the command line or a script.

Each debug view is self-contained, so there is no need to configure any options using the Eclipse main menu.

The debugger's supports multiple simultaneous debug connections. Just launch the debugger several times from the Debug Configurations dialogue.

Some of the debug views, for example the Trace view, are only enabled for JTAG target connections.

Take time to explore the toolbars of each of the views. Press the down arrows  to see more buttons and options.



The screenshot shows the DS-5 Debug perspective in Eclipse IDE. The interface is divided into several panes, each with a specific function. Red callout boxes with arrows point to various features:

- Control target execution with source and assembly level stepping.** Points to the Debug Control toolbar.
- Open, close and manage debugger views.** Points to the Project Explorer.
- Type commands with auto-completion and on-line help.** Points to the Commands pane.
- Drag & drop lines from history view to scripts view to create a script.** Points to the History and Scripts panes.
- Switch between the C/C++ and DS-5 Debug perspectives.** Points to the DS-5 Debug button in the top right.
- Fast View bar for minimizing often-used debug views.** Points to the Fast View bar at the bottom left.
- Synchronized source and assembly code views with color coding.** Points to the Source and Disassembly panes.
- Memory, register, variable and expression views with multiple formatting options.** Points to the Memory, Registers, Variables, and Expressions panes.
- Freeze and replicate any system views in the debugger.** Points to the Outline pane.

The main components visible in the screenshot include:

- Debug Control:** Shows the current target (Pegatron Generic - gdbserver stopped) and execution status.
- Project Explorer:** Displays the project structure, including source files like `main.cpp` and `blockops-noclutter.c`.
- Commands:** A list of commands for debugging, such as `step`, `Execution stopped at breakpoint 1: 0x0`, and `Execution stopped at: 0x0001529C`.
- History:** A list of history entries, including `main() @ main.cpp:43 (gnometris)` and `0x40516368 @ libc.so.6?`.
- Scripts:** A list of scripts, including `BlockOps::rotateBlock()+44 @ blockops-noclutter`.
- Memory:** A table of memory addresses and values, such as `0x0000C184 0xE92D4870 pH-`.
- Registers:** A table of register values, such as `0x0000C188 0xE28DB010 \ M`.
- Variables:** A table of variable values, such as `0x00009E00 const BlockO`.
- Expressions:** A table of expression values, such as `false (0) bool`.
- Outline:** A list of system views, including `config.h`, `blockops-noclutter.h`, `blocks.h`, and `BlockOps::BlockOps()`.

Execution Control and Instruction Trace

Debug Control View

The DS-5 Debugger supports multi-thread applications and multiple simultaneous target connections.

The **debug control view** has a toolbar for processor control operations, which can be used to connect to a target, run, stop, and single-step through code.

By clicking on the debug control view you can refocus the debugger's system views (source, disassembly, variables, registers, outline) on a connection, application thread, kernel thread, or stack frame.

Breakpoints and Watchpoints

Create **new breakpoints** by double clicking on the left column of the source code or disassembly views.

Create **new watchpoints** (data breakpoints) by right-clicking on a cell of the variable or memory views.

Edit the properties of a breakpoint to assign:

- **Condition:** the breakpoint only stops the execution if the condition (a C expression) is evaluated true.
- **Ignore count:** number of times the breakpoint is hit until the application stops.
- **Run script:** the debugger performs an action (e.g log a message) automatically when the breakpoint is hit.
- **Thread-specific breakpoints** stop the processor only when hit from a certain thread.

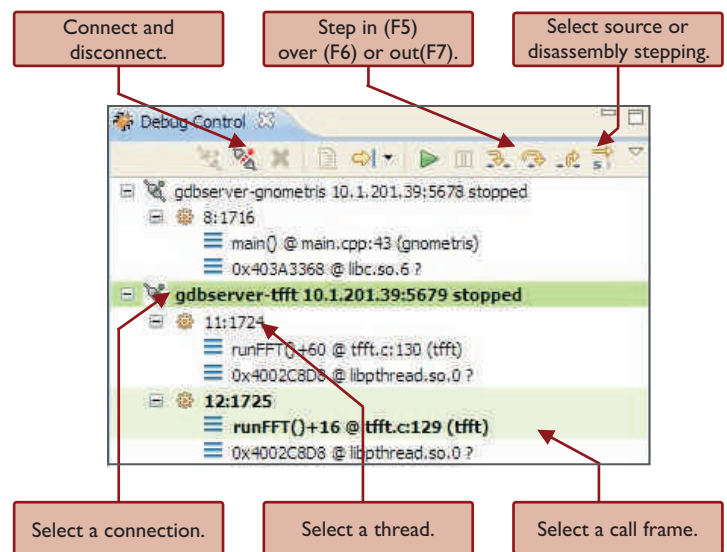
Instruction Trace

Instruction trace enables the non-intrusive debug of random time-related issues, which are difficult to replicate by stepping through the code, and the analysis of performance-critical areas of software.

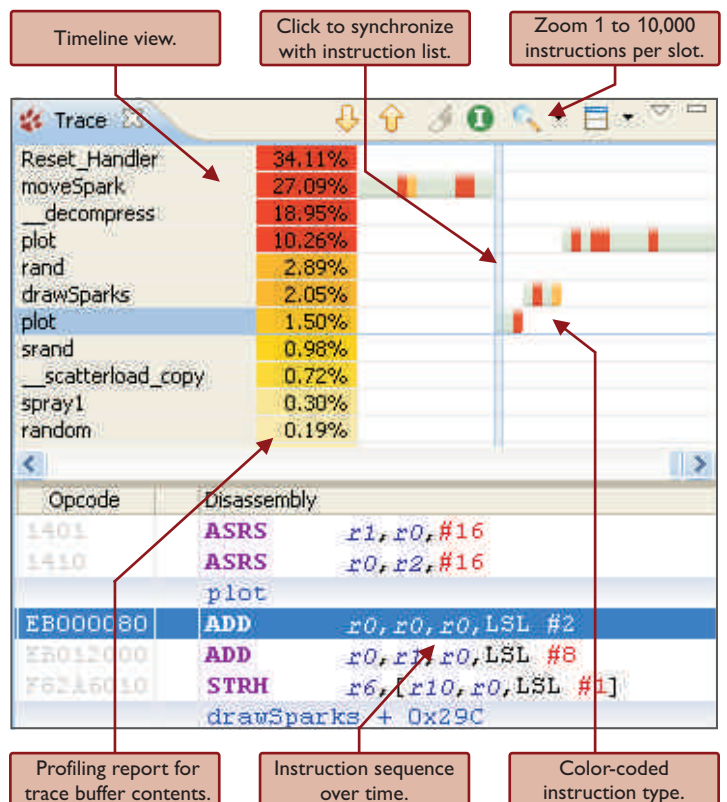
When trace is enabled, the debugger records the last instructions executed by the target. When the execution stops, the debugger fills the trace view with a history of instructions synchronized with the source view.

Trace requires a JTAG connection to devices featuring an **Embedded Trace Macrocell (ETM)** or **Program Trace Macrocell (PTM)**.

The number of instructions recorded is limited by the size of the on-chip **Embedded Trace Buffer (ETB)**. If you need a larger buffer, do off-chip trace with a DSTREAM unit, which includes a 4GB buffer.



Simultaneous debug of **gnometris** and **tfft** Linux applications.



Trace View in disassembly mode.

Debugger System Views

Registers, Variables and Expressions

The **Variables** view shows the value of all the application's variables under three groups: locals, file statics, and globals.

When debugging images with many global variables you can use the **Expressions** view to display the variables of interest. This often makes the debugger faster.

Both views show variable-related information such as type, size, location (register or memory), and value.

You can **drag & drop** variables from the code and variables view to the expressions view. You can also type expressions directly in the expressions view.

When doing bare metal development, use the **Register** view to view the processor, co-processor and peripheral registers with the same format as in the target's documentation.

Using the System Views

The variables, registers, memory, and disassembly views display the status of the system and use color-coding to show what content has recently changed.

All the system views can be **frozen** or **replicated** for easy comparison of the system status at two points in time. They also provide configurable type and data size for display.

Most debugger views accept C/C++ expressions. For example, you can set the value of a register or a memory address with the expression `(*my_ptr) & 0xfffff`.

Open the **screen** view to display the contents of a frame buffer if you are doing LCD driver or graphics development.

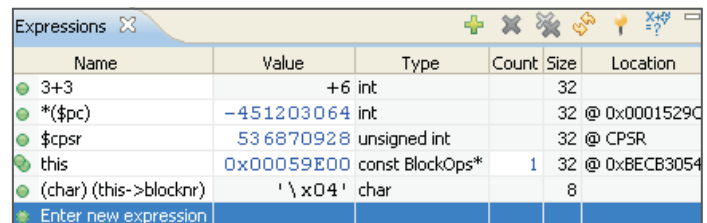
The way the debugger accesses the memory system (e.g., as bytes, words, or multiple words) is configured via the **memory map**. View the memory map with the **info mem** command, and configure it with the **mem** command.

Debugging Disassembly

The debugger uses color-coding to match the contents of the source code and disassembly views. The variables and expressions views show variable-register allocation.

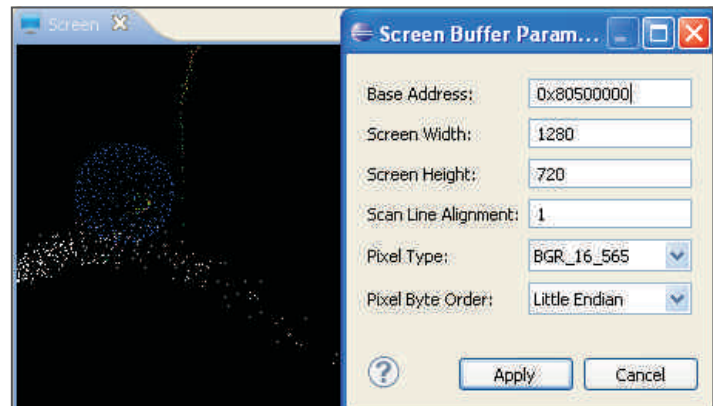
This enables you to easily track how the compiler generates assembly code, even when configured at high levels of optimization.

You can step through code for which debug symbols are not available by manually selecting the disassembly type to ARM or Thumb. This is useful, for example, for boot code debug.

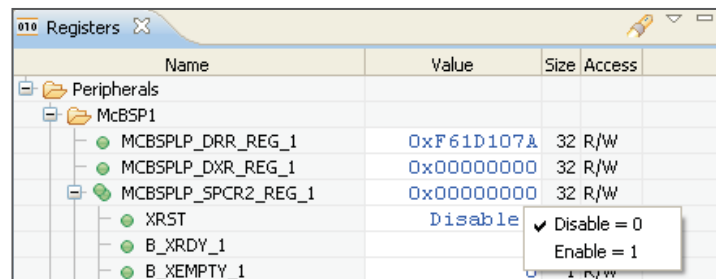


Name	Value	Type	Count	Size	Location
3+3	+6	int		32	
*(\$pc)	-451203064	int		32	@ 0x0001529C
\$cpsr	53 6870928	unsigned int		32	@ CPSR
this	0x00059E00	const BlockOps*	1	32	@ 0xBECB3054
(char) (this->blocknr)	' \ x04 '	char		8	
Enter new expression					

Expressions view.

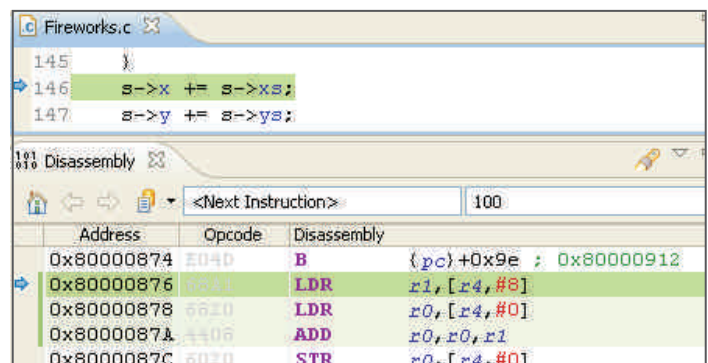


Screen view settings for a TI OMAP3530.



Name	Value	Size	Access
Peripherals			
McBSP1			
MCBSP1_DRR_REG_1	0xF61D107A	32	R/W
MCBSP1_DXR_REG_1	0x00000000	32	R/W
MCBSP1_SPCR2_REG_1	0x00000000	32	R/W
XRST	Disable		✓ Disable = 0 Enable = 1
B_XRDY_1			
B_XEMPTY_1			

Peripheral registers on a TI OMAP3530 device.



Address	Opcode	Disassembly
0x80000874	E040	B {pc}+0x9e ; 0x80000912
0x80000876	8811	LDR r1,[r4,#8]
0x80000878	6610	LDR r0,[r4,#0]
0x8000087A	4406	ADD r0,r0,r1
0x8000087C	6020	STR r0,[r4,#0]

Synchronized source code and disassembly views.

Starting a Performance Analysis Session

Streamline is a sample-based performance analysis tool. It records samples of the program counter, call stack and performance counters, processes the information, and displays it in useful reports.

Setting up your system

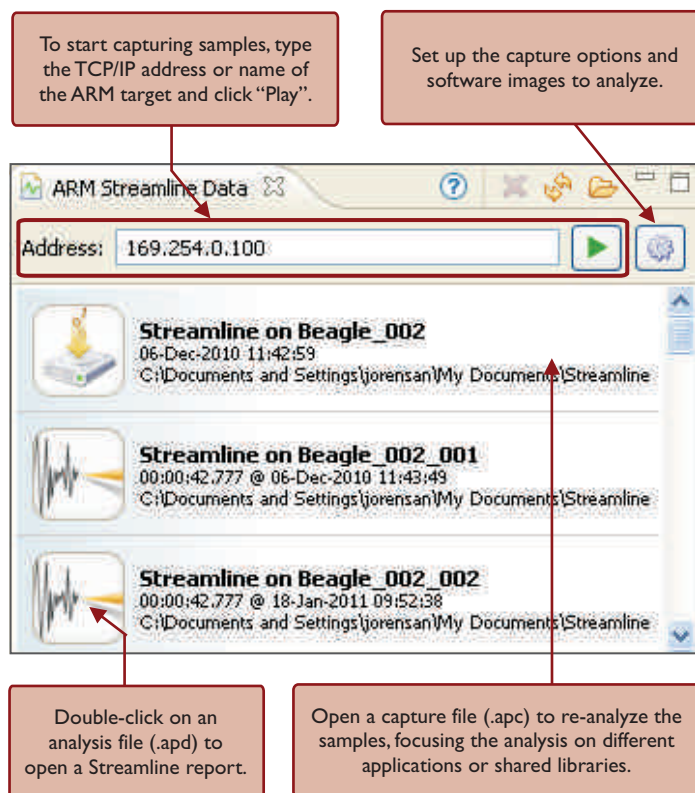
To get the most out of Streamline you need:

- **Network connection** to the target. Streaming over Ethernet enables long capture sessions. Alternatively, store samples on a memory buffer on the target for minimal intrusiveness.
- **Build the Streamline module** with the target's Linux kernel and copy the Streamline daemon to the same directory. Streamline works on Linux 2.6.32 or later.
- **Enable frame pointers** when you build your application and libraries. This enables Streamline to unwind the call stack and generate call-path reports.

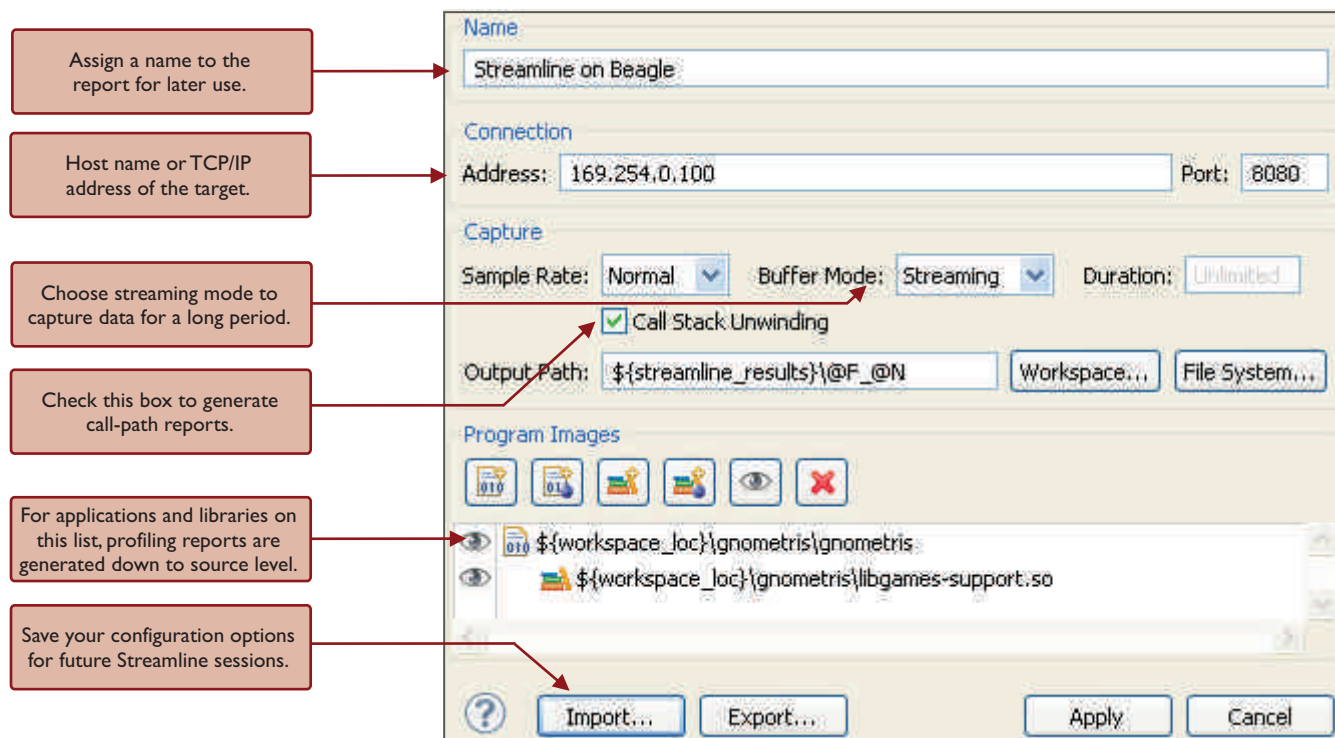
Analyze with libraries. Loading the symbols of libraries and shared objects enhances the analysis of applications.

Launching Streamline

To use **Streamline** open the ARM Streamline Data view. From the Eclipse menu select **Window ► Show View ► Other...** and open the ARM Streamline folder.



The Streamline Data view gives access to previous reports.

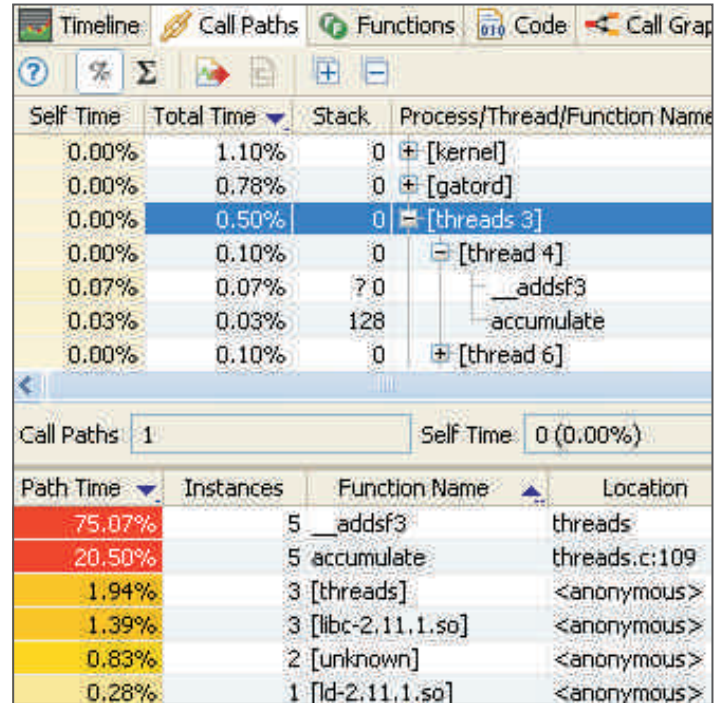
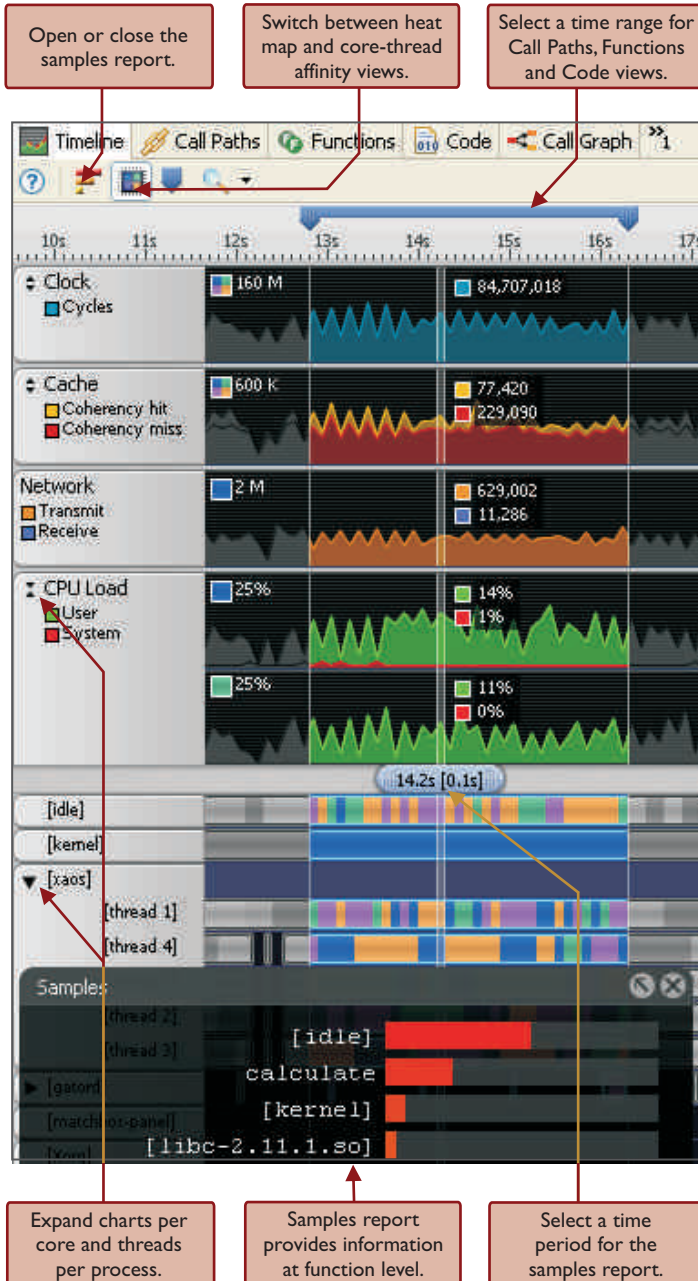


The Streamline Capture Options dialogue enables you to select the right balance between granularity and information detail, and intrusiveness.

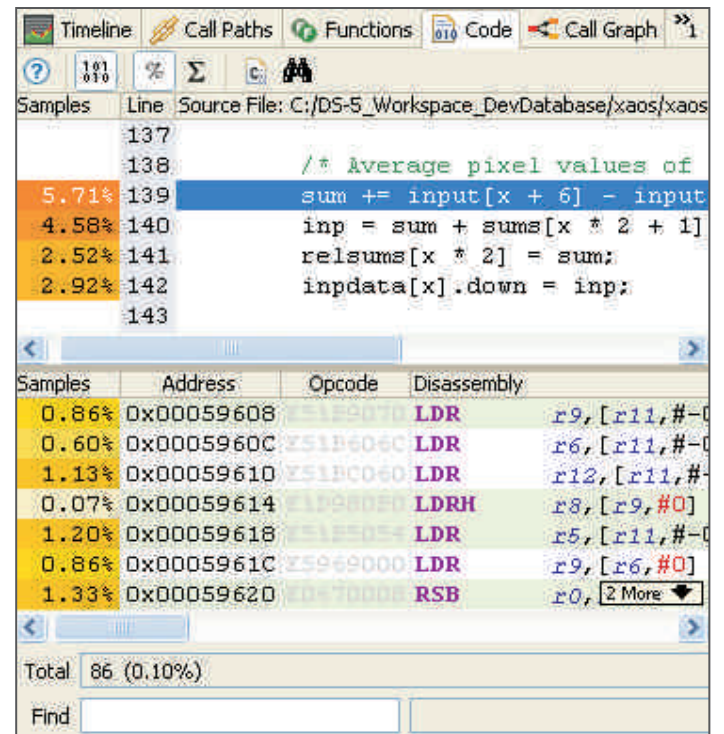
Navigating a Streamline report

A Streamline report provides a great depth of performance-related information on six panes: Timeline, Call Paths, Functions, Code, Call Graph, and Stack Analysis.

Start with the Timeline and Functions for an overview of the execution. Use the other panes to focus your analysis. Use the tabs on the top to navigate between the panes. Alternatively, right-click on a function name to highlight the function on another pane.



Call paths view shows the processor time spent on each call tree. A flat profiling report is generated for the selected call path, which enables you to focus the analysis on a process or thread.



Code view highlights the hot spots within a function by displaying the processor time spent on each line of source code and on each ARM or Thumb disassembly instruction.

Timeline view shows process and thread information over time, matched to SoC performance counters. This enables you to spot thread deadlocks and inefficiencies, as well as hot spots in time.

DSTREAM High-Performance Debug and Trace Unit

DSTREAM

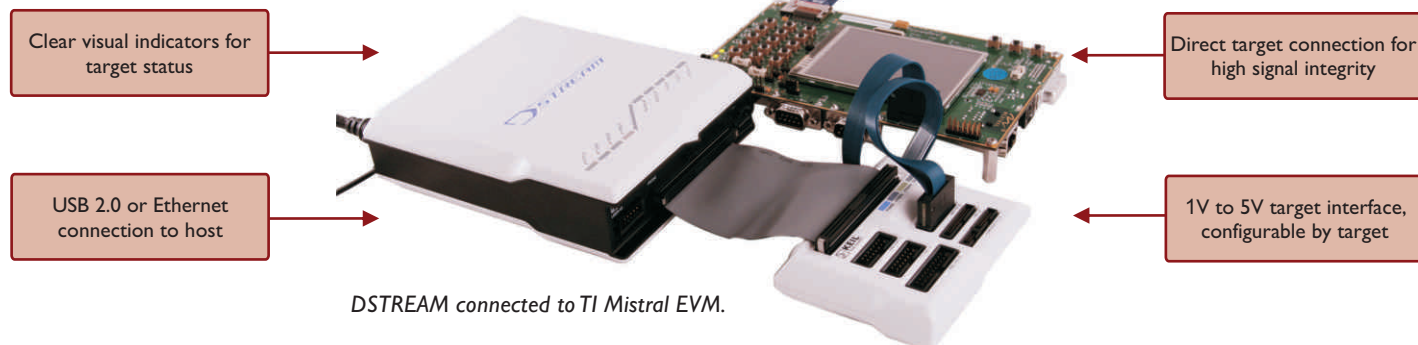
The ARM **DSTREAM** Debug and Trace Unit provides the DS-5 Debugger with high-performance debug and trace connection to hardware targets.

DSTREAM enables DS-5 Debugger to bring up a platform and debug bare metal software in stop-mode, with full access to the processor and system resources.

Thanks to its large buffer DSTREAM supports long-term off-chip trace. This enables the debug of complex, time-related software bugs, and performance optimization of critical code.

	Specification
Debug Interface	JTAG and Serial-Wire Debug
JTAG Speed	60 MHz
Download	2.5 MB/s
Trace Buffer	4GB
Trace Speed	600 Mbps/pin (16-bit)
Trace Frequency	300 MHz DDR
Timestamps	48-bit, 10 ns resolution

DSTREAM features.



DSTREAM connected to TI Mistral EVM.

Find out more about DS-5 and DSTREAM

DS-5 is available in three editions:

	Application Edition	Linux Edition	Professional Edition
Eclipse IDE and Project Manager	✓	✓	✓
GNU Compiler for Linux	✓	✓	✓
Linux Application Debugger	✓	✓	✓
Streamline Performance Analyzer	✓	✓	✓
Real-Time System Models	✓	✓	✓
Boot Code and Driver Debug (JTAG)		✓	✓
Boot Code and Driver Trace		✓	✓
ARM Compiler			✓
Processor Support	Apps cores	Apps cores	Any core

DS-5 Application Edition supports user-space Linux and Android native development with a low-cost package.

DS-5 Linux Edition targets bring-up, driver and application development on Linux systems.

DS-5 Professional Edition extends the functionality of DS-5 to support any ARM-based SoC running bare metal, Linux and RTOS-based software.

Web resources include product download, information and documentation, knowledge articles, support forums and sales contacts. For more information see www.arm.com/ds5 and www.arm.com/dstream.

Please send any feedback on the products or this guide to feedback-ds5@arm.com.



Information in this document is subject to change without notice and does not represent a commitment on the part of ARM. ARM, and Thumb are registered trademarks of ARM Ltd. Cortex, and Keil are trademarks of ARM Ltd. All other brands or product names are the property of their respective holders.